Titre: Title:	New architectures for fast convolutional encoders and threshold decoders
Auteurs: Authors:	David Haccoun, Pierre Lavoie, & Yvon Savaria
Date:	1987
Type:	Rapport / Report
Référence: Citation:	Haccoun, D., Lavoie, P., & Savaria, Y. (1987). New architectures for fast convolutional encoders and threshold decoders. (Technical Report n° EPM-RT-87-46). https://publications.polymtl.ca/10065/

Institution:	École Polytechnique de Montréal
Numéro de rapport: Report number:	EPM-RT-87-46
URL officiel: Official URL:	
Mention légale: Legal notice:	

EPM/RT-87/46

NEW ARCHITECTURES FOR FAST CONVOLUTIONAL ENCODERS AND THRESHOLD DECODERS

DAVID HACCOUN, Ing. Ph.D.
PIERRE LAVOIE, Ing. M.Sc.A.
YVON SAVARIA, Ing. Ph.D.

Ecole Polytechnique de Montréal

Novembre 1987

CA2PQ
UP 5
R87-46

CONSUL

Dépôt légal, 4e trimestre 1987
Bibliothèque nationale du Québec
Bibliothèque nationale du Canada

Pour se procurer une copie de ce document, s'adresser au:

Editions de l'Ecole Polytechnique de Montréal
Ecole Polytechnique de Montréal
Case postale 6079, Succursale A
Montréal (Québec) H3C 3A7
(514) 340-4000

Compter 0,10$ par page (arrondir au dollar le plus près) et ajouter 3,00$ (Canada) pour la couverture, les frais de poste et la manutention. Régler en dollars canadiens par chèque ou mandat-poste au nom de l'Ecole Polytechnique de Montréal. Nous n'honorerons que les commandes accompagnées d'un paiement, sauf s'il y a eu entente préalable dans le cas d'établissements d'enseignement, de sociétés ou d'organismes canadiens.

NEW ARCHITECTURES FOR FAST CONVOLUTIONAL ENCODERS AND THRESHOLD DECODERS*

by

David HACCOUN, Pierre LAVOIE, and Yvon SAVARIA

Department of Electrical Engineering

Ecole Polytechnique de Montreal

C.P. 6079 station A

Montreal, Quebec, Canada

H3C 3A7

* This research was supported in part by the Natural Sciences and Engineering Research Council of Canada, and by the Fonds FCAR of Quebec.

NEW ARCHITECTURES FOR FAST CONVOLUTIONAL ENCODERS AND THRESHOLD DECODERS

by

David HACCOUN, Pierre LAVOIE, and Yvon SAVARIA

ABSTRACT

Several new architectures for high speed convolutional encoders and threshold decoders are developed. In particular, we show that new architectures featuring both parallelism and pipelining are promising from a speed point of view. These architectures are practical to use for a wide range of coding rates and constraint lengths. Two integrated circuits featuring these new architectures have been designed and fabricated in a CMOS 3 micron technology. The two circuits have been tested and can be used to build convolutional encoders and definite threshold decoders operating at data rates above one hundred Mbps. It is shown that with the new architectures, encoders and threshold decoders could easily be designed to operate at data rates above one Gigabit/s.

NEW ARCHITECTURES FOR FAST CONVOLUTIONAL ENCODERS AND THRESHOLD DECODERS

by

David HACCOUN, Pierre LAVOIE, and Yvon SAVARIA

I. INTRODUCTION

In modern digital communications systems and especially in satellite communications, the advantages and potential benefits provided by Forward Error Correction (FEC) are increasingly recognized, and therefore channel coding is becoming an essential element in the design of these systems [1][2]. In FEC systems the main difficulties usually reside in the actual realization of powerful decoders that can operate at high data rates, deliver low error probabilities while being practical and not unreasonably complex to implement. The problem of designing and manufacturing fast encoding and decoding devices at the lowest possible cost is further compounded as the data rates increase, reaching in the hundreds megabits per second (Mbps), and even beyond.

The object of this paper is to present the development of new and fast efficient architectures for implementing convolutional encoders and threshold decoders. *Convolutional codes* are codes that are particularly suitable when the information data to be transmitted arrive serially rather than in packets. A convolutional encoder processes the information in a continuous fashion, a few bits, or even one bit at a time.

Pipeline and parallel architectures for convolutional encoders are developed and their complexity analyzed. High speed encoders can be obtained for any coding rate by combining the two architectures. A one-

chip pipeline encoder realization is presented. This integrated circuit is programmable and operates at 25MHz.

Since any decoder for convolutional codes includes a local encoder, then the encoders developed here may be used in conjunction with any decoding technique, whether powerful and complex such as in Viterbi or sequential decoding, or relatively modest and simple such as in feedback or threshold decoding.

In this paper, only *threshold decoding* [3] is considered. Even though threshold decoding achieves smaller coding gains than those provided by Viterbi or sequential decoding, its main advantage is that the decoder is simple to implement. Furthermore this type of decoder is useful on burst error channels (such as HF, troposcatter, and some telephone channels) since interleaving and de-interleaving can easily be included in the encoder and decoder [4]. Therefore developing efficient architectures for threshold decoders is justified on its own right. Moreover since the actual decoding algorithm is very similar to the encoding process, then selecting this type of decoder and applying to it the architectures developed for the encoder becomes very natural.

A number of factors can further justify the choice of threshold decoding for practical VLSI implementations. For example, the simplicity of the threshold algorithm makes it highly suitable for fast processing. Furthermore, contrarily to Viterbi or sequential decoding, threshold decoding is also convenient to use with high code rates that require a smaller bandwidth expansion (e.g. R=2/3, 3/4, 7/8,...). Finally, even though threshold decoding is far from being optimum in utilizing a convolutional code of given length, it can be used with very long codes (e.g. good codes) that are well beyond the usual range of the more powerful decoding techniques.

Threshold decoder realizations or prototypes, fabricated either as LSI custom devices or from standard parts, are usually designed specifically for a particular code [5]-[9]. By contrast the method for designing threshold decoders presented in this paper is more general and systematic: it even encourages fabricating decoders for very long codes through the use of a single programmable chip as a building block. With the new architectures that are developed in this paper, the pin count required for the programmable chip can be small, and the clock rate of a decoder can become independent of the length of the code used.

The novel architectures feature *pipelining* and *parallelism*, and might be of interest for very high data rate applications. Pipelining occurs in a line of several processors where each processor performs a fraction of the processing and passes partial results to the next one in the line. In a pipelined machine, the data rate is thus no longer limited by the time required to fully process one data element. Parallelism used in conjunction with pipelining allows a further increase of the data rate by increasing the bandwidth of the machine. In practice the cost of this expansion is translated as a greater number of components, or as a larger chip area.

In this paper, each new novel architecture is illustrated by an example showing an implementation for a particular code. For simplicity and with no loss of generality, the same code is used for all the examples. Architectures for encoders are presented in section II, where both the pipeline and the parallel approaches are developed. An actual implementation of a fast encoder that has been designed and fabricated in a CMOS 3 micron technology is described in section III. Preliminary testing shows that an encoder built with a number of these chips could achieve a data rate higher than 150 Mbps. Development of original archi-

tectures for threshold decoding is presented in section IV. Following a description of threshold decoding, a new pipeline architecture for implementing a fast threshold decoder is presented. The modifications required to implement feedback decoding are described. In section V, a parallel architecture for threshold decoding is presented and an actual implementation of a fast threshold decoder is described in section VI. The 5500-transistor chip has been designed and fabricated in the same 3 micron CMOS technology as the encoder. It is programmable and can be used as a building block for more complex and/or faster parallel decoders.

II. A PARALLEL ARCHITECTURE FOR CONVOLUTIONAL ENCODERS

A. Preliminary considerations

A typical non-systematic convolutional encoder of coding rate $R=1/V$ and *memory* M may be represented by a shift register of M delay units connected to V modulo-2 adders [2]. The connections between the shift register and the V modulo-2 adders specify the code. At each clock cycle, one information bit enters the shift register, and the outputs of the V modulo-2 adders are sampled and transmitted in the channel. The V encoded symbols are called a *codeword*, and clearly the coding rate is $R=1/V$ bit/code symbol.

A convolutional code is said to be *systematic* if the first of each V-symbol codeword is the actual information bit that generated that codeword. Consequently in a systematic convolutional encoder of rate $R=1/V$, only $(V-1)$ modulo-2 adders are required, and the V-symbol codeword corresponding to the information bit i_t entering the encoder at

time t consists of i_t and the (V-1) parity symbols, denoted $p_t^{(n)}$, n=1,2,...,(V-1), delivered by the (V-1) modulo-2 adders. A systematic encoder of rate R=1/2, and memory M=6 is shown in figure 1.

A more general encoder of coding rate $R=U/V$ may be viewed as a linear finite state machine receiving U information bits at each clock cycle, and delivering V encoded symbols, $V>U$. A convenient way to implement such an encoder involves U shift registers side by side, connected to the V modulo-2 adders. The memory M of such an encoder is again the sum of all delay units in the encoding machine. For general rate U/V codes the U shift registers may not have all the same length. In order to simplify the design and provide uniform operation it is more convenient to consider all the shift registers to be of the same length L, chosen to be that of the longest shift register. Of course, any delay unit that may be added to extend the length of a shift register to L will not be connected to any of the V modulo-2 adders. The length L is called the *basic length* of the encoder, and for codes of rate $1/V$ this basic length is equal to the constraint length of the code.

Clearly with such an approach the complexity of a general rate U/V encoder varies proportionally to the length L of the encoder rather than the memory M of the code. As for the error performance of convolutional codes, it can be shown that the error probability decreases exponentially as the memory M increases and/or the coding rate R decreases [2].

A straightforward implementation of an encoder as described above leads to problems with implementing the modulo-2 adders, especially if high data rates are required. Indeed a direct implementation of a multi-input modulo-2 adder is not practical and a tree structure of 2-input XORs is usually preferred. Unfortunately, an increase of the number of

inputs on a modulo-2 adder increases with it the depth of the XOR tree. Since the time necessary to compute a parity digit is determined by that depth, the larger the number of inputs on a modulo-2 adder, the slower an implementation inspired by figure 1 will be. Fortunately this problem can be circumvented by the use of a speed-oriented architecture: such an approach is described next.

B. A pipeline architecture

Moving the XORs inside the shift registers leads to pipeline processing of the information digits. In that case a shift register does not shift delayed information digits anymore, but rather partially computes the parity digits. This approach has been proposed by Massey [3]. A pipeline encoder example that implements the same code as in figure 1 is shown in figure 2. Note that the connections must be inverted in order to obtain the same output sequence from the pipeline decoder as from the conventional encoder.

There are several advantages for using this pipeline architecture [10]. First, a pipeline encoder can, at a modest cost, become very easily programmable. Figure 3 shows a convenient arrangement where the connections are loaded serially in a separate register before encoding begins. Moreover, the time required to encode one codeword is independent of the length L of the encoder, and is approximately equal to the delay of a single XOR gate. Therefore an encoder using this architecture can operate at a very high speed, regardless of the memory of the code.

One further advantage is that a pipeline encoder can readily be segmented in several parts, should, for example, a multi-chip implementation be required or preferred. In that case, only a partially computed

parity digit, which is already available at the end of one segment of the encoding pipeline, needs to be transferred to the input of the next segment. Of course this transfer can slow down the encoding process if it takes too long a time. However the encoding speed will not keep on decreasing when the number of segments exceeds 2, since all inter-segment transfers occur concurrently according to the pipeline effect.

A universal pipeline encoder for codes of rates U/V can now be developed. Clearly, since the delayed information digits are no longer available, then a rate $1/V$ encoder requires V separate shift registers to generate V parity digits. Moreover if $U>1$, then U shift registers are necessary for each parity digit. A general pipeline encoder for codes of rate U/V is shown in figure 4.

In practice, the speed of a pipeline encoder will probably be limited not so much by the speed of the encoding circuitry, as by the time necessary to input and retrieve signals from a chip. Furthermore in order to increase further the data rate of the encoder, a wider bandwidth at the input and output of the encoder is necessary. A new parallel architecture which can provide a speed increase is described next.

C. A parallel architecture

The encoding architecture presented above allows only one codeword to be generated per clock cycle. It is however possible, with an original parallel architecture, to design an encoder capable of generating an arbitrary number of codewords at each clock cycle. The following simple example illustrates this idea.

Figure 5 shows our usual encoder of figure 1 at two different times, denoted t and t+1. Since the coding rate is equal to 1/2, each

codeword is two digits long and, obviously, two clock cycles are thus necessary to generate the codewords (i_t, p_t) and (i_{t+1}, p_{t+1}).

By contrast, figure 6 shows a parallel realization of the encoder in figure 5, using the same code and producing identical output sequences. Note that at each clock cycle, the parallel encoder accepts two consecutive information digits and delivers the two corresponding codewords. Therefore, for the same clock frequency, the parallel encoder generates the codewords at a data rate twice that of the conventional encoder.

For simplicity, but without loss of generality, the parallel encoder of figure 6 was limited to two codewords per clock cycle. However, for all coding rates, there exists a parallel encoder that can generate an arbitrarily large number Y of codewords per clock cycle. Y is called the *parallelism coefficient*. Furthermore, the parallel and pipeline architectures can be combined to produce a fast parallel programmable encoder.

The number of components and interconnections of a device is referred to as the *area complexity* and denoted A. Similarly the amount of time T required to produce a certain amount of processing is called the *time complexity*. The *overall complexity* AT is thus the product of both the area and the time complexities. In the evaluation of T, the machine is assumed to be synchronous, or in other words, the propagation delay on wires is assumed not to increase significantly with the size of the machine. Using shift registers all of equal basic length L the complexities of the programmable parallel-pipeline encoding architecture for non-systematic convolutional codes are given by:

$$A = O(LUVY) \tag{1}$$

$$T = O((UY)^{-1}) \tag{2}$$

Note that while the number of encoding pipelines is proportional to Y^2, the length of each pipeline is approximately shortened by a coefficient $1/Y$, which explains why A is only proportional to Y. Finally the use of a systematic code would reduce the area complexity of the encoder to:

$$A = O(LU(V-U)Y) \qquad (3)$$

Note that the area complexity grows only linearly with the length L of the code, and that the time complexity is not only independent of that length, but inversely proportional to U. The most striking result, however, is that the overall complexity AT is independent of the parallelism coefficient Y, and therefore it is possible through parallelism to trade efficiently the area for a higher data rate.

III. IMPLEMENTATION OF A FAST CONVOLUTIONAL ENCODER

A fast one-chip programmable encoder has been designed, fabricated and tested [10]. It implements a basic building block composed of six encoding cells. Many such chips can be connected together to implement a rate $1/V$ encoder for long memory codes. Furthermore, if external XOR circuits are supplied, any code of rate U/V can be generated. The chip can also be used to build a parallel-pipeline encoder for very high data rates.

The chip has been fabricated in a 3 micron CMOS technology with one metallization and one polysilicon layers. It has been tested and showed to encode correctly a random input sequence at clock frequencies up to 25 MHz, implying that the design is probably I/O limited. For example, at that frequency and using a parallelism coefficient Y=3, rates 2/3 and 3/4 encoders could operate at 2 X 3 X 25 = 150 Mbps, and 3

X 3 X 25 = 225 Mbps respectively.

The length of the encoding pipeline on the test chip has been voluntarily limited to 6. Since this particular implementation features parallel loading of the shift register's contents, the number of pins required grows rapidly. However, this feature is not essential and, as can be seen on figure 7, the encoding circuitry covers a very small silicon area. In fact, with the same 3 micron technology, a larger 9 X 9 mm^2 die could contain a programmable pipeline encoder of length L=2000.

Pipeline encoding chips could readily be fabricated in a faster CMOS technology and operate at clock frequencies above 100 MHz. Increasing the coding rate or using parallelism could then bring forth encoder realizations with data rates above 1Gbps. It should be pointed out however that the *VY* output digits exiting the encoding pipelines must be time-multiplexed if they are destined to be transmitted over a serial channel. Fortunately 1.5 Gbps multiplexers and demultiplexers are readily available [11]. The architecture for high speed encoding suggested here appears to be very cost-effective since all the encoding process is implemented in a cheap CMOS technology, and only the multiplexers components need to be fabricated in an expensive high-speed technology such as GaAs.

IV. A PIPELINE ARCHITECTURE FOR THRESHOLD DECODING

A. General description of threshold decoding

The encoders described above may be used in any decoder implementation for convolutional codes, whether complex and powerful (Viterbi, sequential) or simple and modest (feedback, threshold), since regardless of the decoding technique, a decoder always requires a local encoder. Of the many decoding algorithms for convolutional codes, threshold decoding is the least complex, and consequently the easiest to implement. Furthermore a very close similarity exists between the actual threshold decoding algorithm and convolutional encoding. Therefore the architecture developed above to improve the speed and complexity of the encoder may be directly applicable to a threshold decoder.

Threshold decoding is based on the concept of an orthogonal parity check and has been first proposed by Massey [3]. Assuming systematic codes of rate $R=1/2$ and memory M, at any time t an estimate $\hat{p}_t$ of the transmitted parity digit p_t is constructed by encoding the corresponding received information bit $i_{r,t}$. A syndrome digit s_t is then formed by adding (modulo-2) the estimated parity digit $\hat{p}_t$ to the corresponding received parity digit $p_{r,t}$. M such syndrome digits are accumulated in a shift register properly connected to a majority logic unit. A number J, $J<M$, of syndrome digits are used by that unit to compute an estimate $\hat{n}_{t-M}$ of the noise digit at time $(t-M)$, which added to $i_{r,t-M}$ yields an estimate of the information bit $\hat{i}_{t-M}$. This estimate $\hat{i}_{t-M}$ is accepted as the decoded information bit at time $(t-M)$ and is delivered to the user. The operation is illustrated in figure 8 where the communication system has been divided in three parts: a convolutional encoder, a memo-

ryless noisy channel and a threshold decoder. For simplicity the channel has been divided in two sub-channels for the separate transmission of the information digit i_t and the parity digit p_t, and the same systematic rate 1/2, memory 6 code illustrated in figure 1 has been used.

Threshold decoding is applicable only with codes featuring special algebraic properties. For simplicity only the so-called *self-orthogonal* codes [3] will be considered here in the development of architectures for threshold decoders. A large number of good self-orthogonal codes are known and tabulated [3][5][6][12]. These codes are widely used in practice, in particular in Single Channel per Carrier systems used for voice transmission over satellite, where a rate 3/4 self-orthogonal code is the standard used by INTELSAT [1].

The threshold decoder shown in figure 8 is said to be *definite* [13]. A definite decoder becomes a *feedback* decoder when the noise estimate $\hat{n}_{t-M}$ is properly fed back in the syndrome register to improve the reliability of the syndrome digits. Figure 9 shows how to add the feedback to the decoder of figure 8, where the syndrome digits of the feedback decoder are denoted s^* instead of s. It can be shown that the error propagation phenomenon is limited when self-orthogonal codes are used [14], and in general feedback decoding is more powerful than definite decoding [3].

B. A pipeline architecture

This section presents a new pipeline architecture for the implementation of a fast threshold decoder. The two major components of the decoder are the encoder and the majority logic unit, and both can operate in a pipeline fashion.

A pipeline implementation of the encoder has been presented above. That same encoder can perfectly fit into the decoder. However we recall that the decoder in figure 8 uses the delayed information digit $i_{r,t-6}$. Since the pipeline encoder does not shift the information digits, one additional shift register must be provided to make this delayed information digit available.

The majority logic unit can be implemented as a J-input arithmetic adder followed by a comparator connected to an appropriate threshold value. However, that configuration suffers from the same problems that led us to consider pipelining for the encoding process: increasing J slows down the computation. Fortunately, pipelining can also be used here, and the speed of the majority logic unit becomes then independent of L and proportional to the logarithm of $(J+1)$. Pipelining the majority logic unit, however, is costly. Indeed the sum-of-syndromes (SOS) pipeline must be at least $\log_2(J+1)$ bits wide in order to process properly the partially computed sums of syndromes. Figure 10 shows in detail a pipeline majority for the same code used in earlier illustrations.

C. Generalization for codes of high and low coding rates

The architecture just introduced can be adapted for codes of rates different from 1/2. The codes most widely used have a rate $(V-1)/V$ or $1/V$, $V=2,3,\ldots$ The adaptation for these rates is presented below.

For systematic codes of rate $(V-1)/V$ (e.g. $R=3/4$, 4/5, ...) each codeword contains only one parity digit. Hence there is only one syndrome, and thus comparisons with the threshold values can be done right after each individual SOS pipeline. In this particular case the pipeline decoder is easy to build as shown in figure 11 for an $R=3/4$ code.

Generalizing the pipeline architecture for codes of rate $1/V$ is not immediate since the majority logic unit accesses $(V-1)$ syndrome registers instead of only one. A satisfying approach is to use a separate SOS pipeline for each syndrome, and to compare to a single threshold value the sum of the $(V-1)$ SOS pipeline outputs. Figure 12 shows a decoder featuring this solution for a code of rate 1/3.

The architectures for codes of rate $(V-1)/V$ and $1/V$ can be combined to yield a general rate U/V pipeline decoder. Now, in order to improve the error performance of the pipeline decoder, we examine incorporating feedback in its architecture.

D. Feedback decoding

While the addition of feedback to the conventional architecture is straightforward, such an addition is not an easy task with the pipeline architecture. It is natural to think that the feedback should be incorporated in the SOS pipeline; that approach has been selected. However a closer analysis reveals that the cost of the feedback addition may be substantial.

In the SOS pipeline, each column contains a partial sum of syndromes digits. Since each syndrome digit can be either 0 or 1, the sum can be any positive number smaller, or equal to the number of parity-check syndrome digits J. At each tapped column, the noise estimate $\hat{n}_{t-(L-1)}$ must cancel the effect of $n_{t-(L-1)}$ in only one particular term of the sum, the *target-syndrome* of that column. Assuming $\hat{n}_{t-(L-1)}$ equals to 1, it is the value 0 or 1 of the target-syndrome that will determine if the sum must be increased by 1 or decreased by 1. So each tapped column of the SOS pipeline needs to access not only the value of

$\hat{n}_{t-(L-1)}$, but also the value of its target-syndrome. Unfortunately there are several different target-syndromes and their distribution along the columns of the SOS pipeline is irregular. This distribution is a function of the code implemented and cannot be changed.

Basically, implementing the feedback involves three modifications to the pipeline majority. First, the SOS pipeline must be redesigned to allow in each column the possibility of adding the values -1, 0, +1, or +2 to the partial sum of syndromes according to several signals. A global view of the new SOS pipeline and the truth table of one column processor are shown in figure 13. Note that in that particular example, every column except the right-most one is tapped, but this is not usually the case, even for optimum codes. Next, a circuit must be added to store the individual syndromes $s^*_t, \ldots, s^*_{t-(L-1)}$ and update them according to $\hat{n}_{t-(L-1)}$. This is readily done using the circuit shown in figure 9. The third modification is the addition of a network allowing each tapped column of the SOS pipeline to access the value of its own target-syndrome. This is clearly a very costly modification if the network must be programmable: every column must then be free to select its own target-syndrome out of $(J-1)$ syndromes. In a non-programmable implementation the network becomes much simpler: a $(J-1)$ digit-wide bus carries the syndromes values along the pipeline and the target-syndrome input of every tapped column is permanently connected to the proper line of the bus.

V. A PARALLEL ARCHITECTURE FOR THRESHOLD DECODING

Exploiting the close similarities between the threshold decoder and the convolutional encoder leads to examine whether parallelism could also be used to increase the data rate of the decoding process. It will be shown with an example again, that indeed it is possible to build fast decoders that match the data rates of the fast encoders described earlier.

Since it is no longer necessary to consider the parallel-only decoding architecture first in order to understand the more interesting parallel-pipeline architecture, we will proceed with that architecture directly. Figure 14 shows a parallel-pipeline decoder for our usual code. Since the decoder takes two codewords and delivers two decoded digits at each clock cycle, it can be twice as fast as a pipeline-only decoder. Of course a decoder could be designed to process any number Y of codewords simultaneously.

The complexities of the parallel-pipeline architecture are:

$$A = O(L \ln((J/Y)+1)\ U(V-U)Y) \tag{4}$$

$$T = O(\ln((J/Y)+1)\ (UY)^{-1}) \tag{5}$$

The factor $\ln((J/Y)+1))$ appearing in every complexity measure describes the variation of the SOS pipeline width as a function of J and Y. We recall from section IV.B that for the pipeline-only architecture an SOS pipeline featuring J taps had to be at least $\log_2(J+1)$ bits wide in order to properly process the partial sums of syndromes. When parallelism is included, however, we have seen that the SOS pipelines are shortened approximately by a factor Y and thus each SOS pipeline fea-

tures on the average, J/Y taps instead of J. The width of each pipeline can thus be reduced on the average to $\log_2((J/Y)+1)$.

Observing from the codes given by Martin [6] and Wu [12] that J is always small in comparison with L, it follows that the area complexity A of the decoder grows almost linearly with the length L of the code: the parallel-pipeline architecture is thus practical even for very long codes. The time complexity, however, is a function of J. Further refinement of the architecture by the use of a two-dimensional pipeline in the SOS logic could yield a time complexity that is independent of J, but would require a larger silicon area. In practice the use of such an architecture may not be justified. For example, in many implementations the speed of a pipeline device is not limited by its internal processing time, but rather by the time required for the signals to enter or exit the device.

Finally, a properly programmed parallel decoder will generate, for a given input sequence, an identical output sequence as a non-parallel decoder. The only difference is that the parallel decoder takes VY channel digits at a time and delivers UY decoded information digits simultaneously. Consequently, the use of parallelism in a decoder implementation involves no additional constraint on the encoder architecture and *vice versa*.

VI. VLSI IMPLEMENTATION OF A FAST THRESHOLD DECODER

A complete R=1/2, L=40 one-chip definite threshold decoder has been designed and manufactured with the same 3 micron CMOS process that has been used for the encoder fabrication [15]. The 5500-transistor chip fits on a 2250 X 4507 square micron die shown in figure 15. The code, majority connections and threshold values can be loaded through a scan chain prior to operation. Most of the area of the chip is covered by a 5-bit wide SOS pipeline.

Many chips can be connected end to end to increase L but the number of taps on the whole SOS structure must not exceed $(2^5-1)=31$. That number seems sufficient since, for example, a rate 1/2 code with J=30 features a length L=841. In future implementations, the width of the SOS pipeline could easily be increased should the need arises. Decoders for codes of rate U/V or parallel decoders can be built with a number of chips, or rows of chips connected side by side. However for rates different from 1/2, 2/3, 3/4, or when parallelism is used, a few external logic circuits must be supplied.

The full-custom chip has been designed by students of a project-oriented VLSI graduate course at *Ecole Polytechnique de Montreal*. The design methodology was basically inspired by the structured approach of Mead and Conway [16]. Since the SOS pipeline is the most area-consuming and time-critical part of the chip, it deserves special attention.

Each cell of the SOS pipeline consists of a dynamic eight-transistor flip-flop, a three-transistor pseudo-nMOS XNOR gate, a NAND gate, and an inverter. The logic and transistor diagrams of the cell are shown in figure 16. The right input at the top of each row is the connection input and must be set to *Gnd* if there is a connection and to *Vdd*

otherwise. The cell measuring only 62 X 210 square microns has been carefully optimized for compactness and speed. By a natural coincidence, the cell can be used to build the encoding pipeline without any modification.

Power buses and clock lines run on the single metal layer to reduce voltage drops and propagation delays. SPICE simulations led to expect a 20 MHz clock frequency, meaning for example, that the data rate of a R=3/4 decoder featuring Y=2 parallelism would be 3 X 2 X 20 = 120 Mbps. Such a decoder would require at least 3 X 2^2 = 12 rows of L/40 X 1/2 = L/80 chips each.

Testing the chip unveiled two minor routing bugs. The errors occurred at the layout step of the design process. Fortunately individual sections of the chip could be tested individually even if the chip itself did not work properly. In particular the SOS pipeline could be partially tested and showed to perform as expected. The two bugs have now been fixed and the chip has been re-submitted for fabrication. When new samples become available, they will be tested to confirm our speed expectations.

At this point it would be very interesting to compare an hypothetical threshold decoder using the parallel-pipeline architecture with a decoder recently implemented in Japan using the soft-decision Viterbi decoding algorithm [17]. This particular one-chip Viterbi decoder uses a code with constraint length K=7, and provides a coding gain of 4.7 db at a bit-error-rate of 10^{-5}. It contains 121000 transistors and operates at a typical data rate of 20 Mbps.

With the same number of transistors (121000), calculations show that it would have been possible to fabricate a programmable feedback threshold decoder for a self-orthogonal code of length K=841, with a

coefficient of parallelism Y=2. Based on our computer simulations, at the bit-error-rate of 10^{-5} such a decoder would yield a coding gain of 2.7db, that is 2 db lower than the coding gain of the Viterbi decoder. Most of that difference can be attributed to the fact that the threshold decoder processes hard-quantized channel symbols whereas the Viterbi decoder uses soft-quantization.

If the data rate of the Viterbi decoder is limited by the operating speed of its add-compare-select modules, the pipeline architecture of the threshold decoder, in turn, would rather be limited by the small delay necessary to enter and retrieve signals from the VLSI chip. Since the technology used for fabricating the Viterbi decoder would certainly allow operation at a clock rate of 100MHz, the threshold decoder featuring a coefficient of parallelism Y=2 would deliver decoded information bits at a data rate of 200Mbps, that is one order of magnitude faster than that provided by the Viterbi decoder, above. Therefore the architectures presented in this paper appear to be attractive principally for fabricating very fast decoders for applications requiring moderate coding gains.

VIII. CONCLUSION

Motivated by a growing demand for efficient FEC systems operating at high data rates, several new architectures for convolutional encoders have been developed. In particular, new architectures featuring both parallelism and pipelining are promising from a speed point of view.

Indeed the data rates of encoders featuring these architectures are not influenced by the memory length of the implemented code. Furthermore, we have shown that, for a given technology, the data rate of

an encoder could be arbitrarily increased by the parallelism factor *Y*. Remarkably this data rate increase appears to be quite affordable since it involves an area enlargement which is only proportional to the same factor *Y*. Finally, we have shown that the new architectures could be used together with any high or low rates *U/V* codes.

Encoders designed with the new architectures might be used in any decoders for convolutional codes, whether powerful (as Viterbi or sequential decoders), or mode modest (as threshold decoders). Moreover, because the threshold decoder shares many similarities with the encoder, pipelining and parallelism can be used not only in its local encoder, but also in the actual decoding circuitry. The architectures developed for the encoder can thus be used also in the design of fast and affordable threshold decoders. The simplicity of these decoders makes threshold decoding an attractive alternative to Viterbi or sequential decoding in some applications.

Two integrated circuits, an encoder and a threshold decoder, have been designed, fabricated and tested to determine the cost of implementing those architectures in a CMOS 3 micron technology. A noteworthy feature of both designs is that the circuits can be used as building blocks to construct encoders or decoders for long codes of any coding rate, and using any parallelism factor *Y*. Since the encoding and decoding logics do not use a large silicon area, it follows that encoders or decoders for codes of very large memory length (e.g. 1000) could be built with only a few integrated circuits. Finally we also showed that CMOS encoders and decoders with a sufficiently large parallelism factor *Y* could operate at very high data rates, in the Gigabits/second range. These new architectures appear to be very attractive for high speed FEC realizations applicable in both terrestrial and space communications systems.

ACKNOWLEDGMENT

The authors wish to thank Jean-Didier Allegrucci and Abdel-Karim Gadiri, graduate students at Ecole Polytechnique, for their important contribution in the design of the two integrated circuits. These circuits have been developed on equipment provided by the Canadian Microelectronics Corporation, and fabricated by Northern Telecom Electronics Ltd. of Ottawa, Canada.

REFERENCES

[1] W.W. Wu, D. Haccoun, R. Peile, and Y. Hirata, "Coding for Satellite Communication", *IEEE Journal on Sel. Areas in Comm.*, vol. SAC-5, no. 4, May 1987, pp. 724-748.

[2] V.K. Bhargava, D. Haccoun, R. Matyas, and P. Nuspl, *Digital communications by satellite*, Wiley, New York, 1981.

[3] J.L. Massey, *Threshold decoding*, M.I.T. Press, Cambridge, Mass., 1963.

[4] J.A. Heller, "Feedback Decoding of Convolutional Codes", *Advances in Communication Systems*, vol. 4, A. Viterbi Ed., Academic Press, New York, 1975, pp. 261-278.

[5] W.W. Wu, "New convolutional codes - part I", *IEEE Trans. on Comm.*, vol. COM-23, no. 9, September 1975, pp. 942-955.

[6] G.D. Martin, "Optimal convolutional self-orthogonal codes with an application to digital radio", *Proc. IEEE Int. Conference on Comm.*, 1985, pp. 1249-1253.

[7] M. Kavehrad, "Convolutional Coding for High-Speed Microwave Radio Communications", *AT&T Technical Journal*, vol. 64, no. 7, September 1985, pp. 1625-1637.

[8] "LSI Data Communication Device Technology Available for Commercial and Industrial Applications", *Computer Design*, vol. 15, no. 12, December 1976, pp. 98-101.

[9] M. Kavehrad, "Implementation of a self-orthogonal convolutional code used in satellite communications", *Electronic Circuits and Systems*, vol. 3, no. 3, May 1979, pp. 134-138.

[10] Y. Savaria, D. Haccoun, and P. Lavoie, "Design tradeoffs for the VLSI implementation of a fast universal convolutional encoder", *Proc. 13th Biennial Symposium on Communications*, Queen's University, Kingston, 1986, pp. A.5.1-A.5.4.

[11] R. Hickling, J. Kemps, and K. Rousseau, "1.5-Gbit/s GaAs multiplexer and demultiplexer forge fast fiber-optic links", *Electronic Design*, vol. 34, no. 2, January 23, 1986, pp. 107-112.

[12] W.W. Wu, "New convolutional codes - part III", *IEEE Trans. on Comm.*, vol. COM-24, no. 9, September 1976, 946-955.

[13] J. P. Robinson, "Error Propagation and Definite Decoding of Convolutional Codes", *IEEE Trans. on Inf. Theory*, vol. IT-14, no. 1, January 1968, pp. 121-128.

[14] J. P. Robinson, "A Class of Binary Recurrent Codes with Limited Error Propagation", *IEEE Trans. on Inf. Theory*, vol. IT-13, no. 1, January 1967, pp. 106-113.

[15] P. Lavoie, Y. Savaria, D. Haccoun, J.-D. Allegrucci, and A.-K. Gadiri, "Implementation of a high speed threshold decoder for long constraint length convolutional codes", *Proc. 1986 Canadian Conference on VLSI*, Montreal, 1986, pp. 139-144.

[16] C. Mead, and L. Conway, *Introduction to VLSI systems*, Addison-Wesley, Reading, Mass., 1980.

[17] T. Ishitani, K. Tansho, N. Miyahara, S. Kubota, and S. Kato, "A Scarce-State-Transition Viterbi-Decoder VLSI for Bit Error Correction", *IEEE Jour. of Solid-State Circ.*, vol. SC-22, no. 4, August 1987, pp. 575-581.

FIGURE CAPTIONS

Fig. 1. A convolutional encoder for a systematic rate 1/2, memory 6 code.

Fig. 2. A pipeline encoder for the same code as in figure 1. The speed of the encoder is no longer related to the number of taps on the register. However the register contains partially computed parity digits instead of delayed information bits.

Fig. 3. A programmable pipeline encoder for the same code as in figure 1.

Fig. 4. A general pipeline encoder for rates $R=U/V$ codes.

Fig. 5. The encoder of figure 1 shown at two different times.

Fig. 6. An $Y=2$ parallel encoder for the systematic rate 1/2, memory 6 code. For the same input sequence it produces exactly the same output sequence as the encoder of figure 1.

Fig. 7. Photomicrograph of a pipeline encoder of basic length 6. This encoder operated as expected at a 25 MHz clock frequency.

Fig. 8. A complete encoder/decoder system for the systematic rate 1/2, memory 6 code, using a definite threshold decoder.

Fig. 9. Addition of feedback in the syndrome register of the decoder illustrated in figure 8.

Fig. 10a. A detailed view of the pipeline majority for the decoder illustrated in figure 8. Each column of the SOS pipeline contains a partially computed sum of syndromes. Since each sum can take a value from 0 to 4, each column must be three cells high to process the binary representation of a sum with no risk of overflow.

Fig. 10b. Logic diagram of one basic cell of the SOS pipeline.

Fig. 11. An R=3/4 pipeline threshold decoder. It can be viewed as three R=1/2 pipeline threshold decoders (in dotted boxes) sharing one modulo-2 adder.

Fig. 12. An R=1/3 pipeline threshold decoder.

Fig. 13a. An SOS pipeline including feedback for the usual systematic rate 1/2, memory 6 code.

Fig. 13b. One processor of the pipeline where i, j, and k are inputs for the connection, the target-syndrome value, and the noise estimate respectively.

Fig. 13c. Truth table of the processor.

Fig. 14. An Y=2 parallel-pipeline threshold decoder for the usual rate 1/2, memory 6 code. The parallelism factor Y can be increased indefinitely, allowing even faster decoders.

Fig. 15. Photomicrograph of an L=40 pipeline threshold decoder.

Fig. 16a. Logic diagram of one basic cell of the SOS pipeline.

Fig. 16b. Transistor diagram of the same cell.

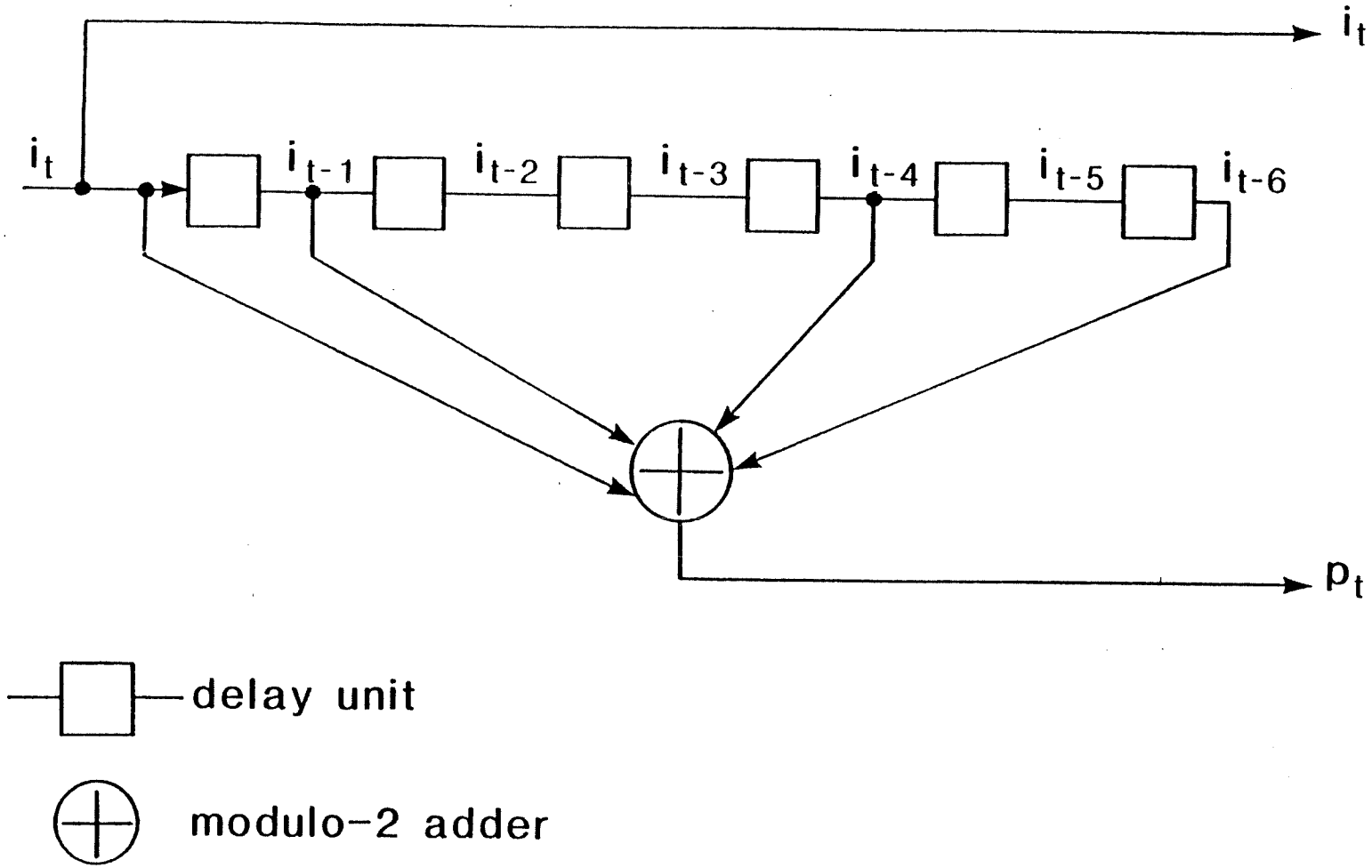


Fig. 1. A convolutional encoder for a systematic rate 1/2, memory 6 code.

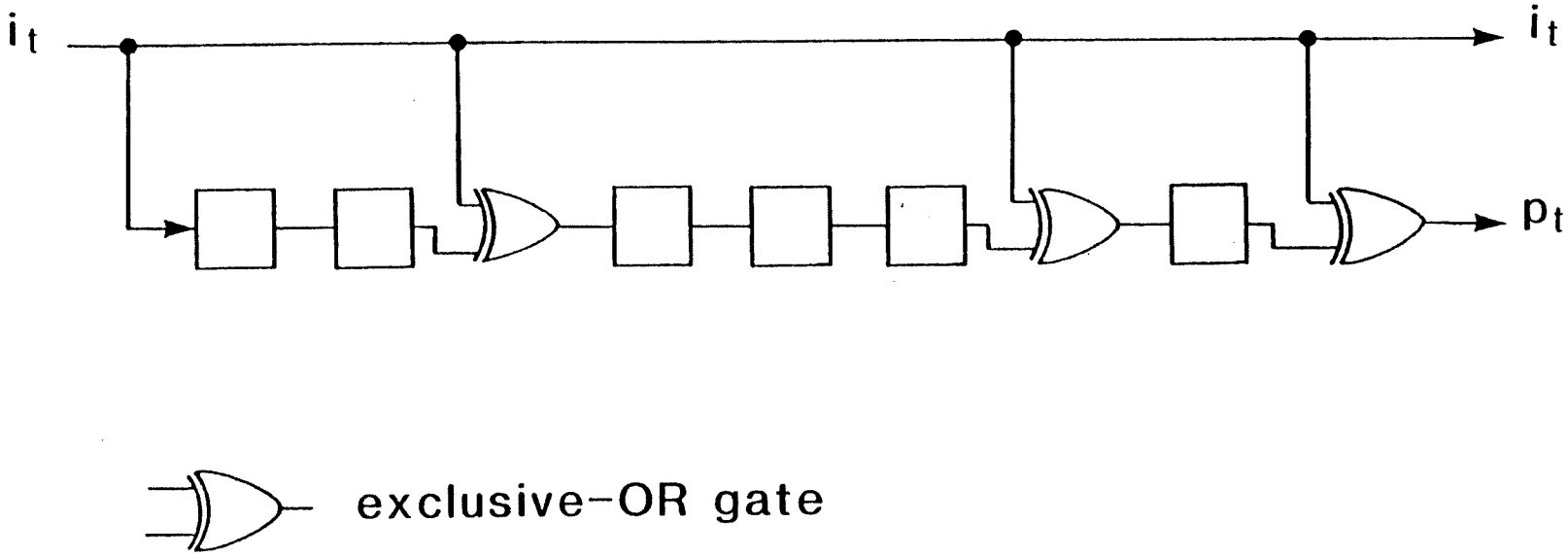


Fig. 2. A pipeline encoder for the same code as in figure 1. The speed of the encoder is not related any more to the number of taps on the register. However the register contains partially computed parity digits instead of delayed information bits.

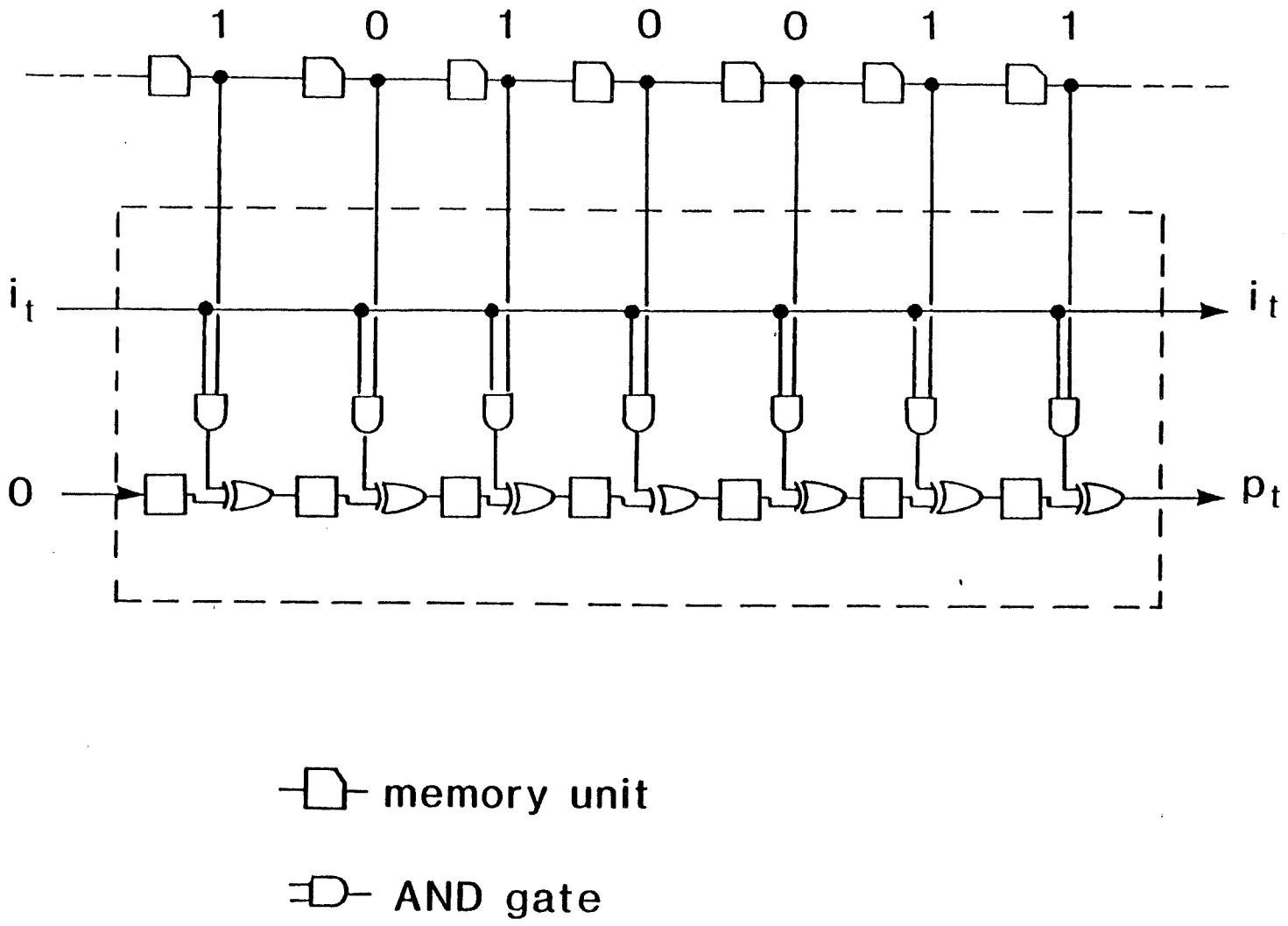


Fig. 3. A programmable pipeline encoder for the same code as in figure 1.

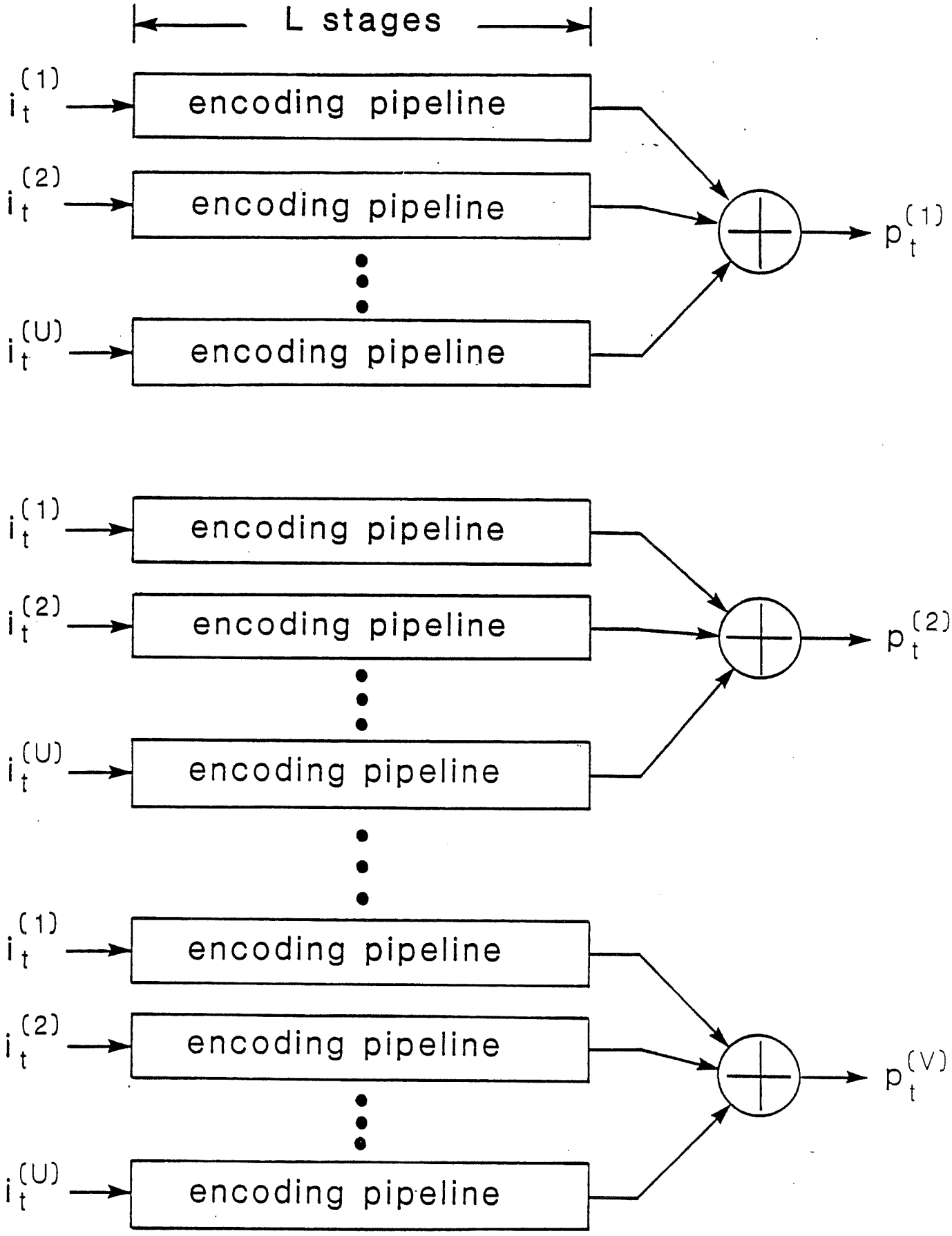


Fig. 4. A general pipeline encoder for $R=U/V$ codes.

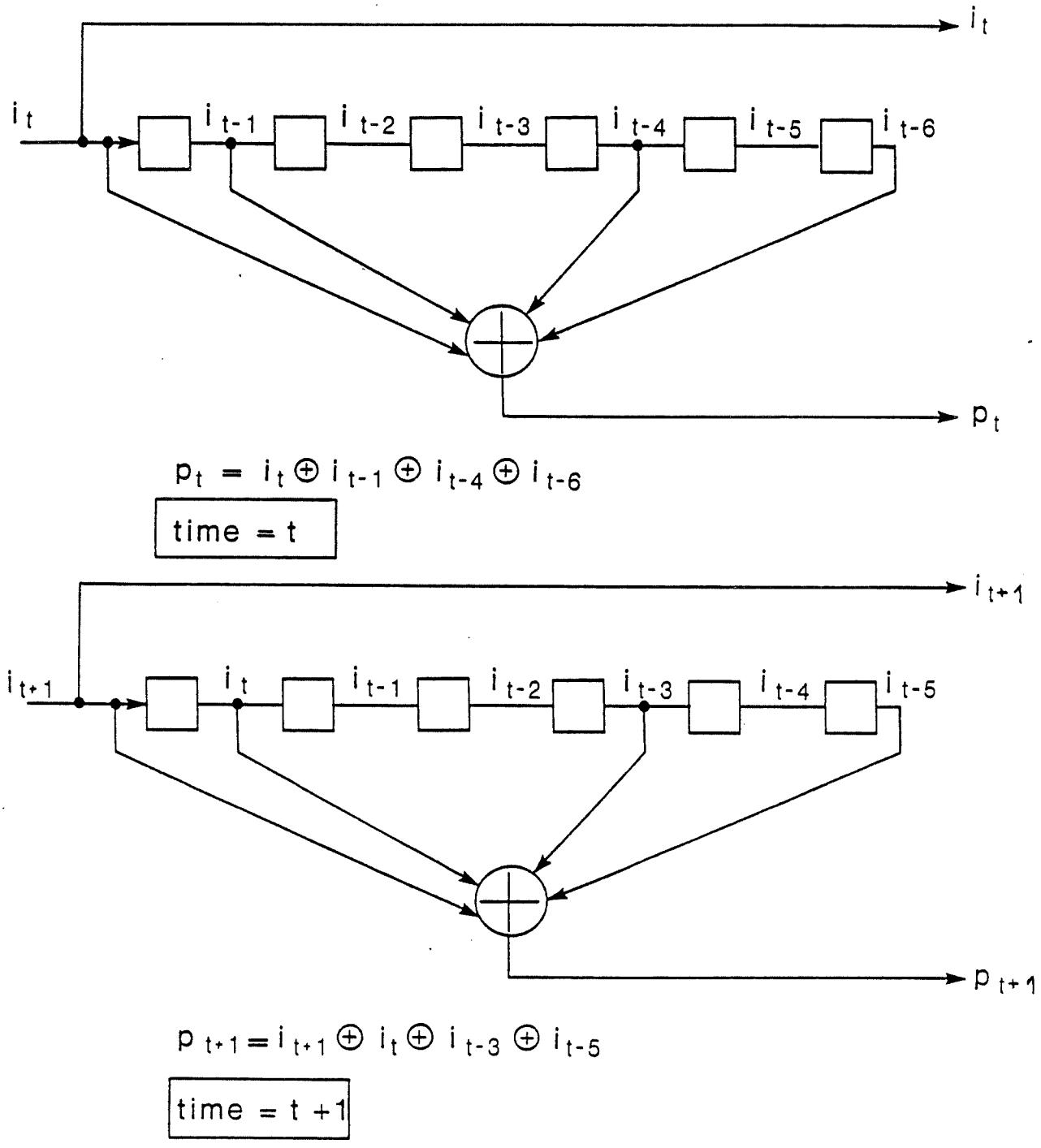


Fig. 5. The encoder of figure 1 shown at two different times.

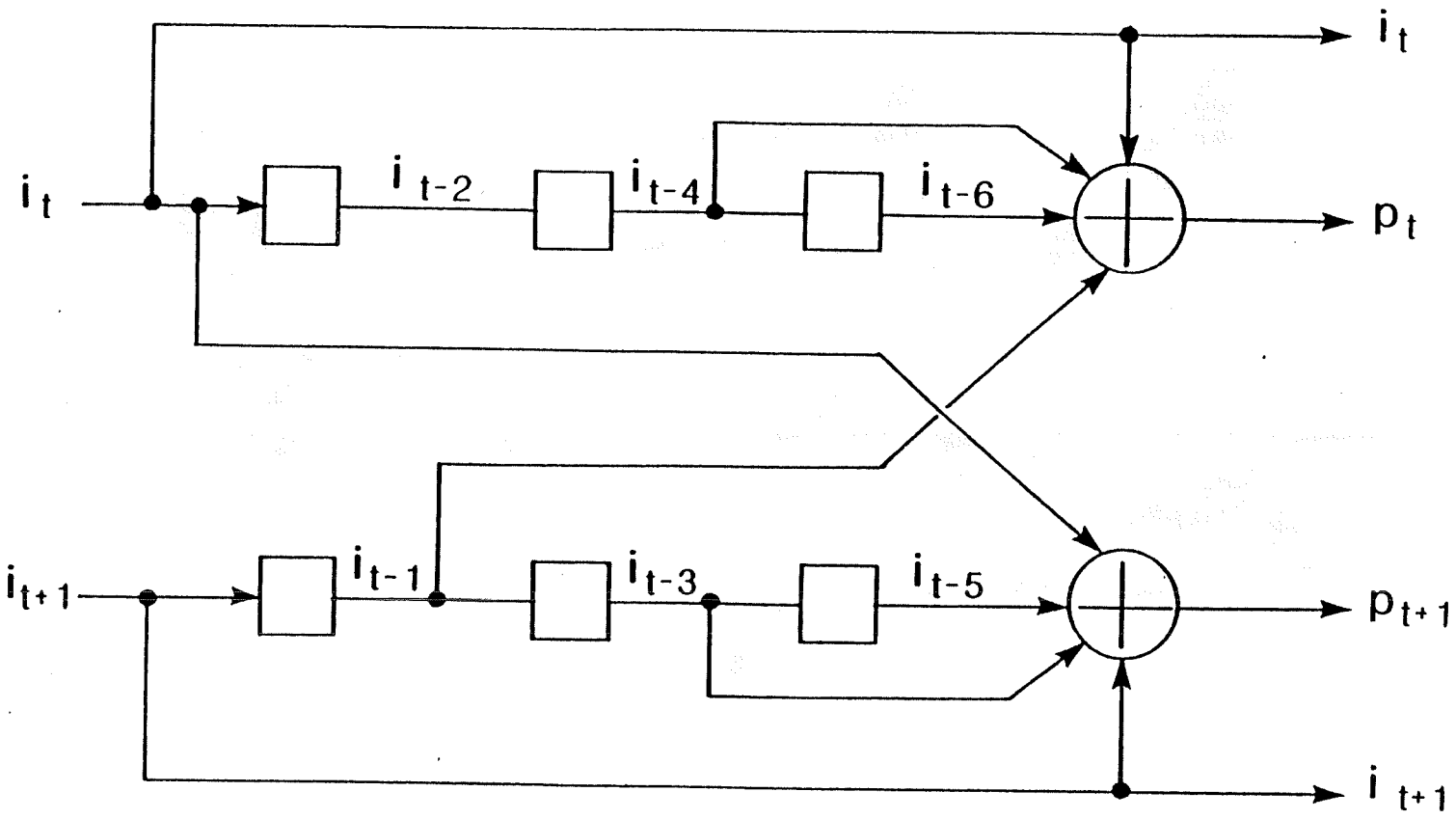


Fig. 6. An $Y=2$ parallel encoder for the systematic rate 1/2, memory 6 code. For the same input sequence it will produce exactly the same output sequence as the encoder of figure 1.

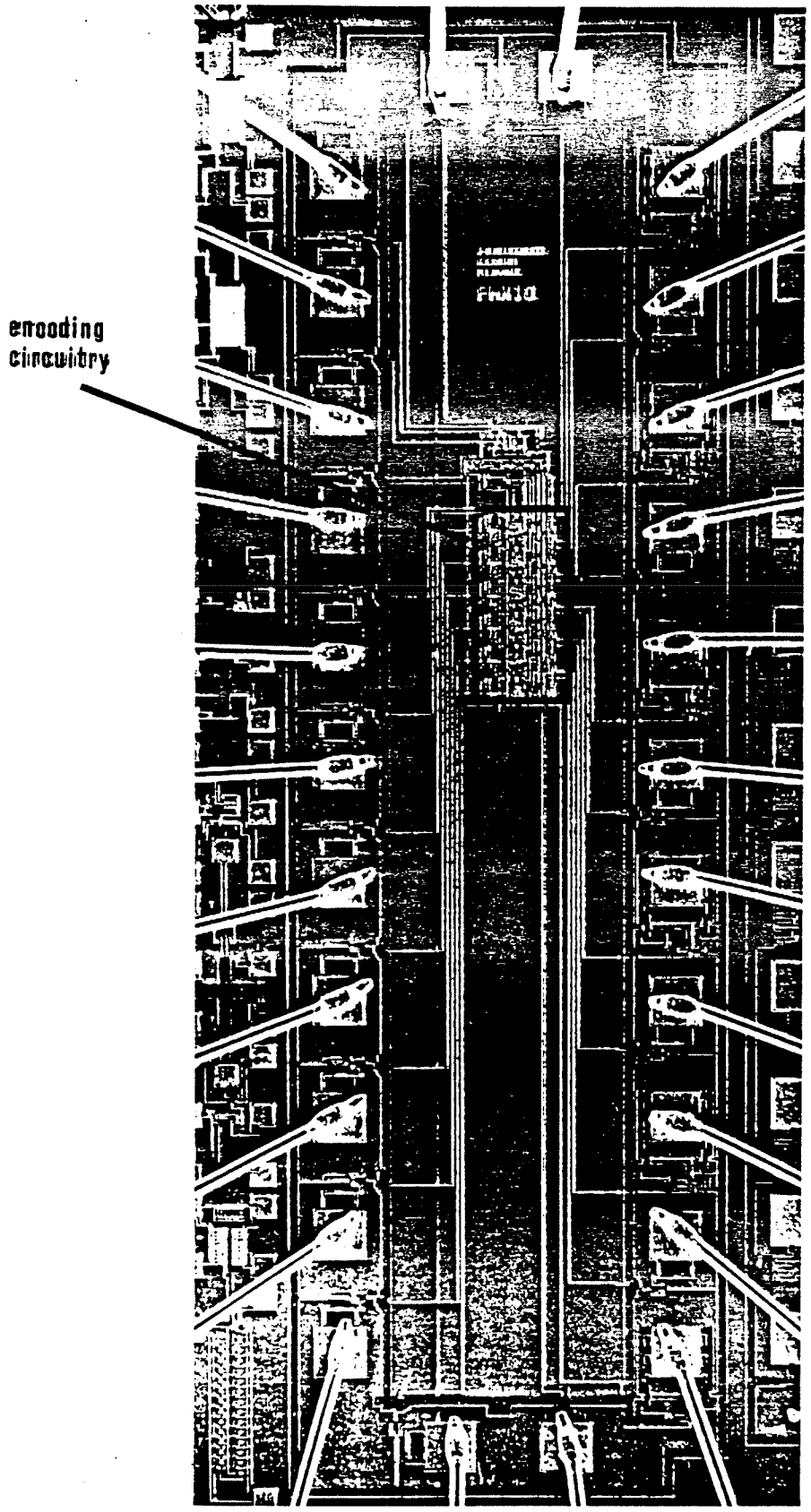


Fig. 7. Photomicrograph of a pipeline encoder of basic length 6. This encoder operated as-expected at a 25 MHz clock frequency.

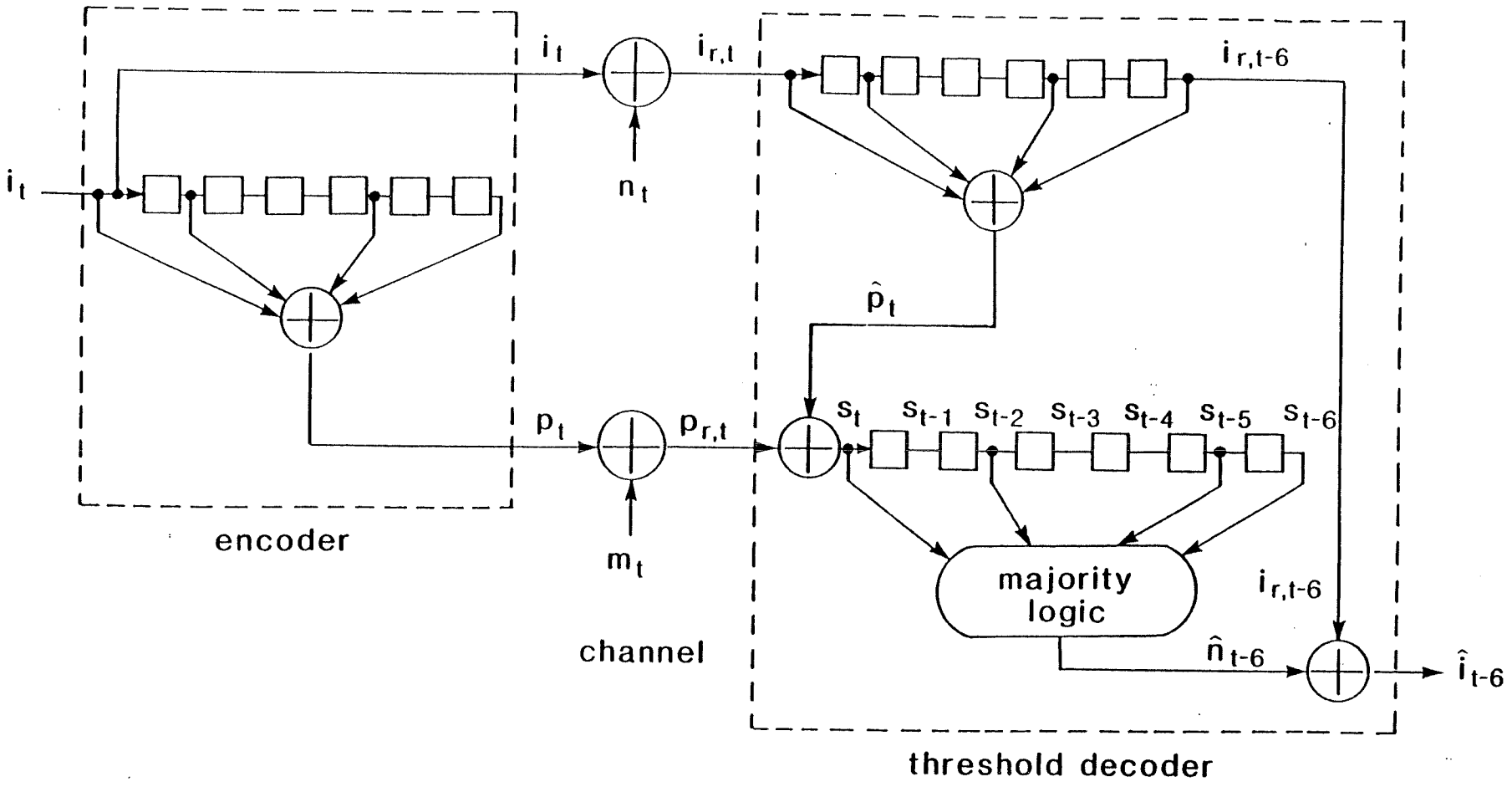


Fig. 8. A complete encoder/decoder system for the systematic rate 1/2, memory 6 code, using a definite threshold decoder.

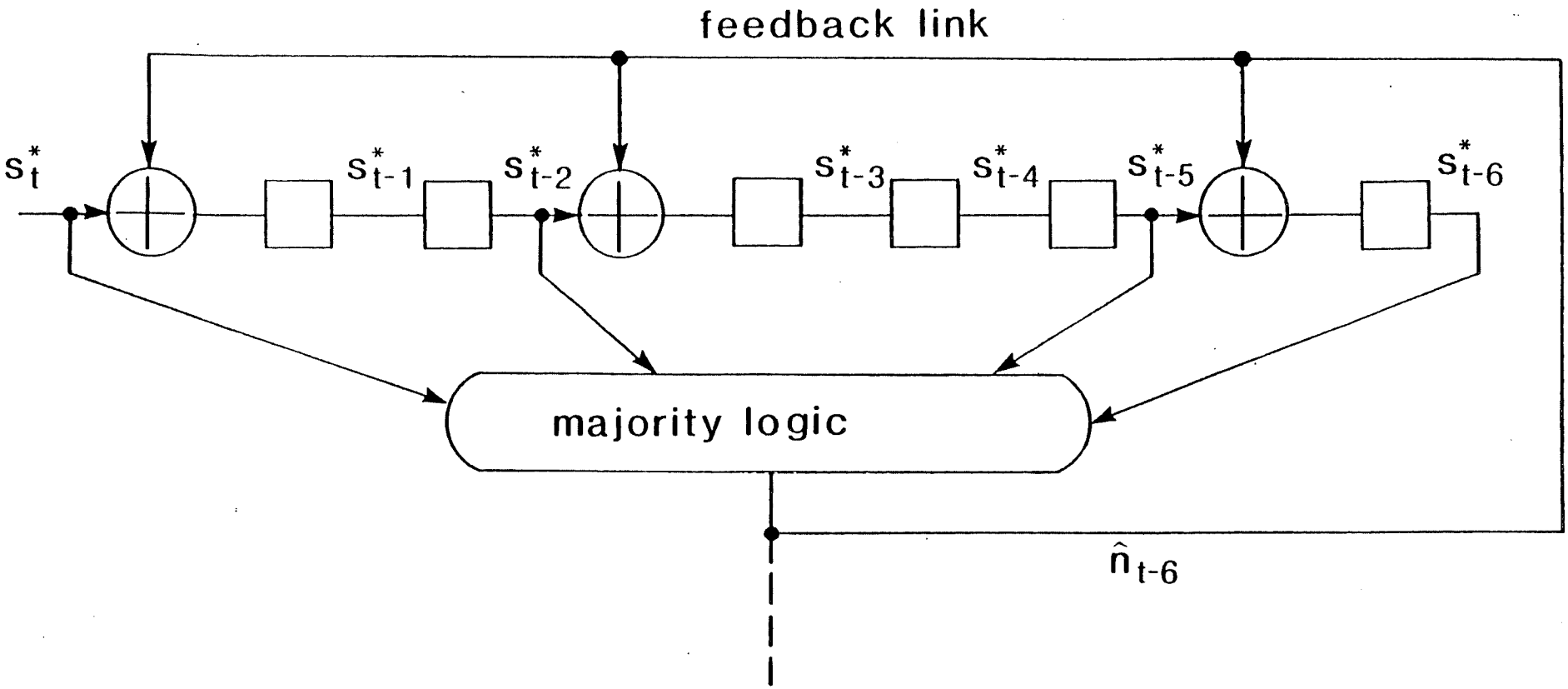


Fig. 9. Addition of feedback in the syndrome register of the decoder illustrated in figure 8.

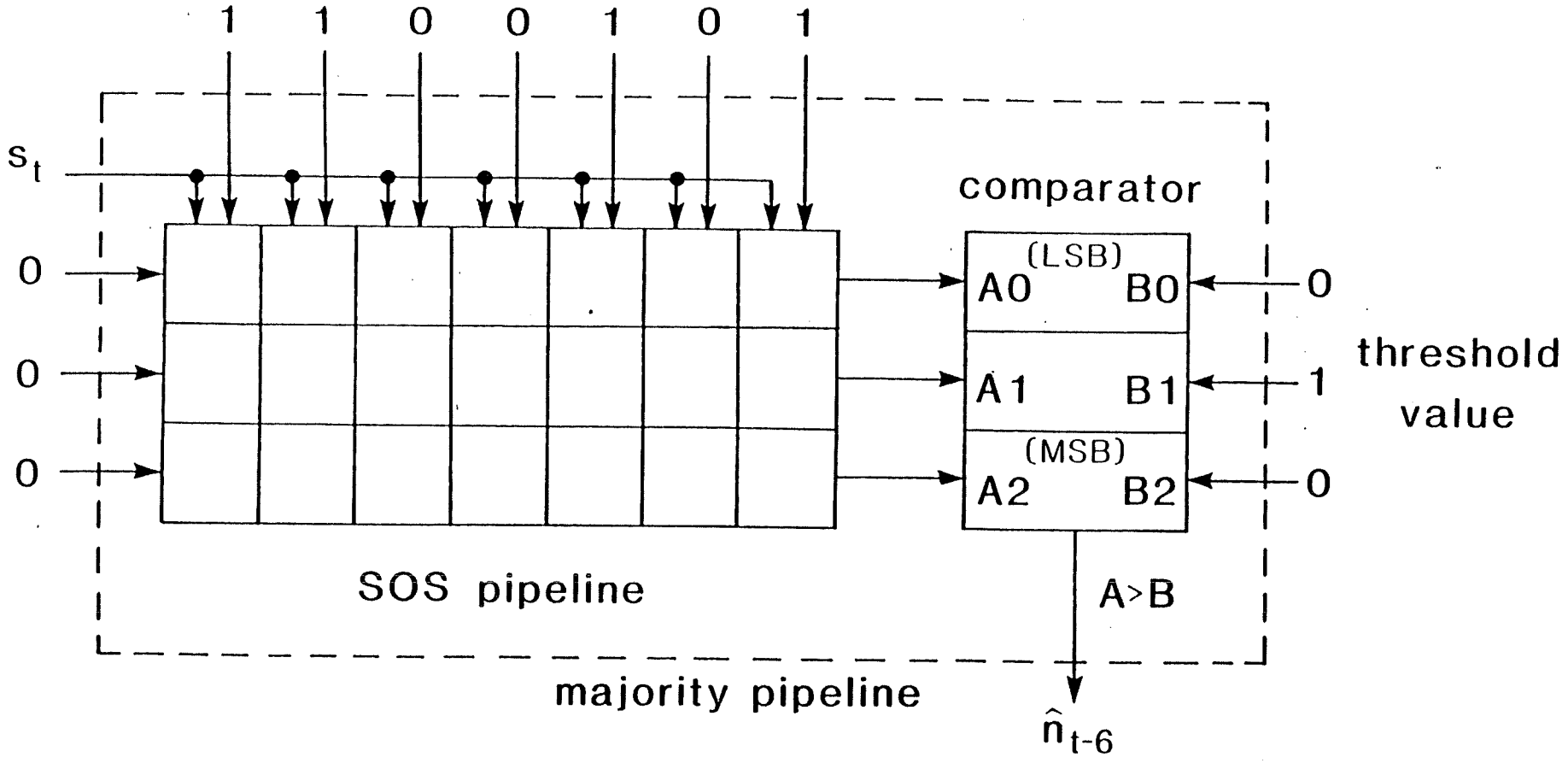


Fig. 10a. A detailed view of the pipeline majority for the decoder illustrated in figure 8. Each column of the SOS pipeline contains a partially computed sum of syndromes. Since each sum can take a value from 0 to 4, each column must be three cells high to process the binary representation of a sum with no risk of overflow.

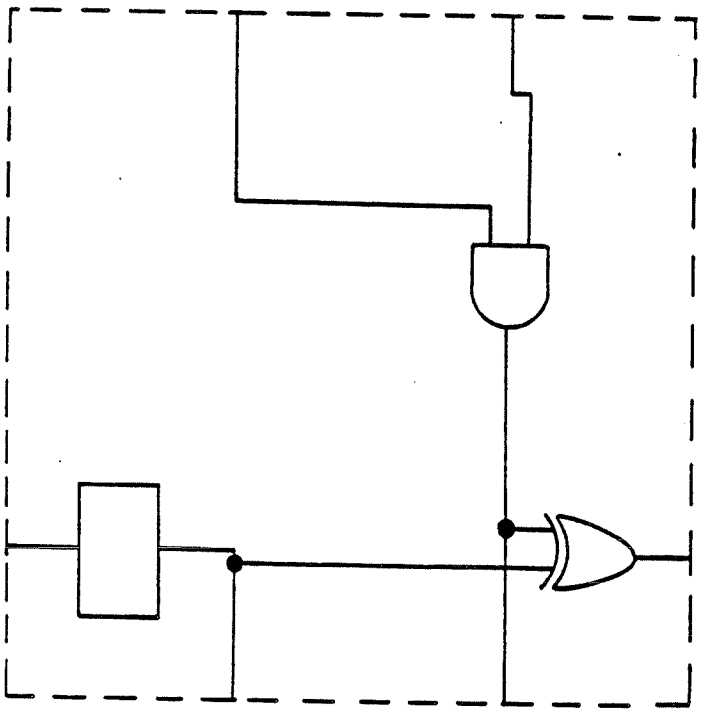


Fig. 10b. Logic diagram of one basic cell of the SOS pipeline.

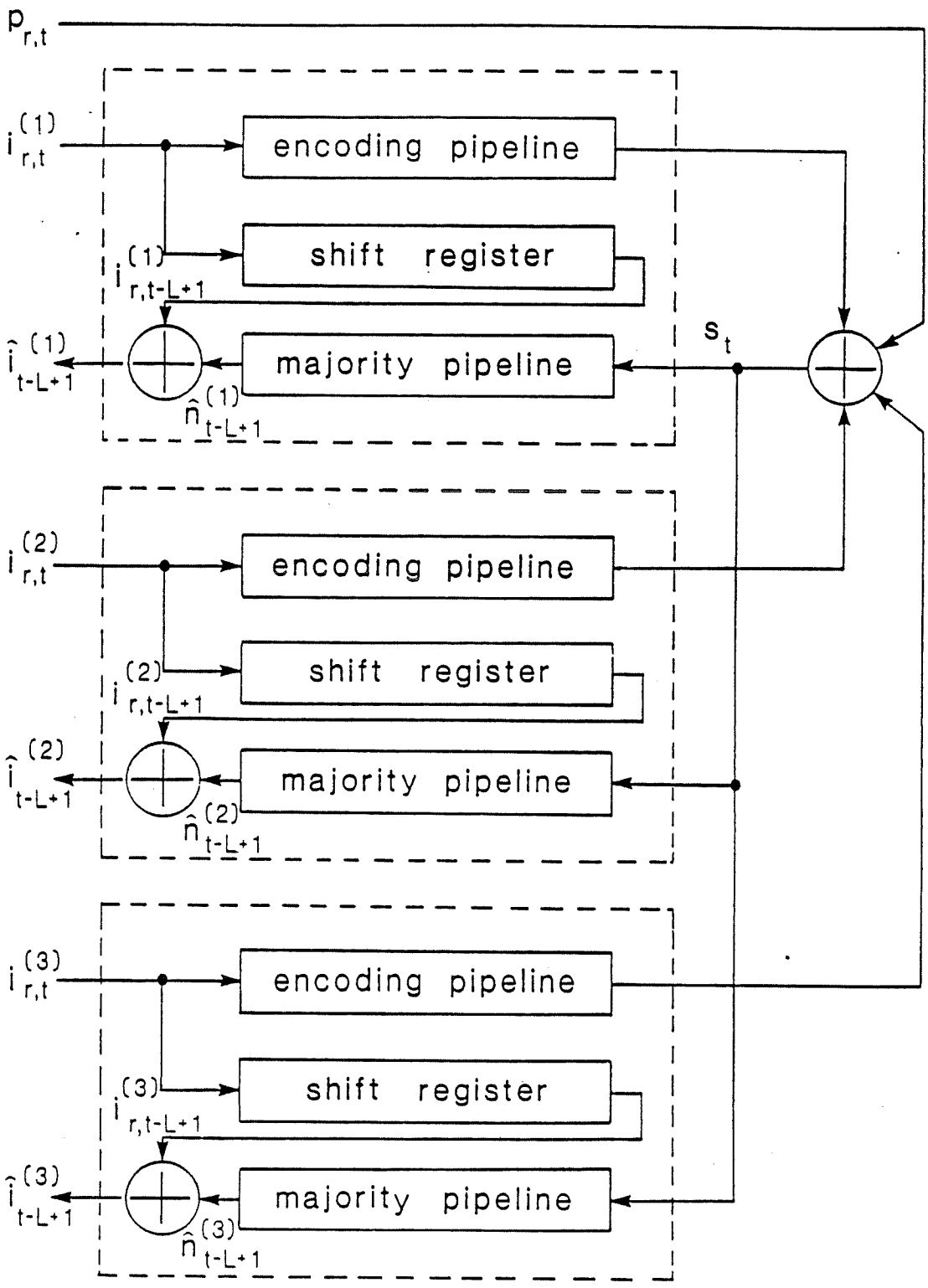


Fig. 11. An R=3/4 pipeline threshold decoder. It can be viewed as three R=1/2 pipeline threshold decoders (in dotted boxes) sharing one modulo-2 adder.

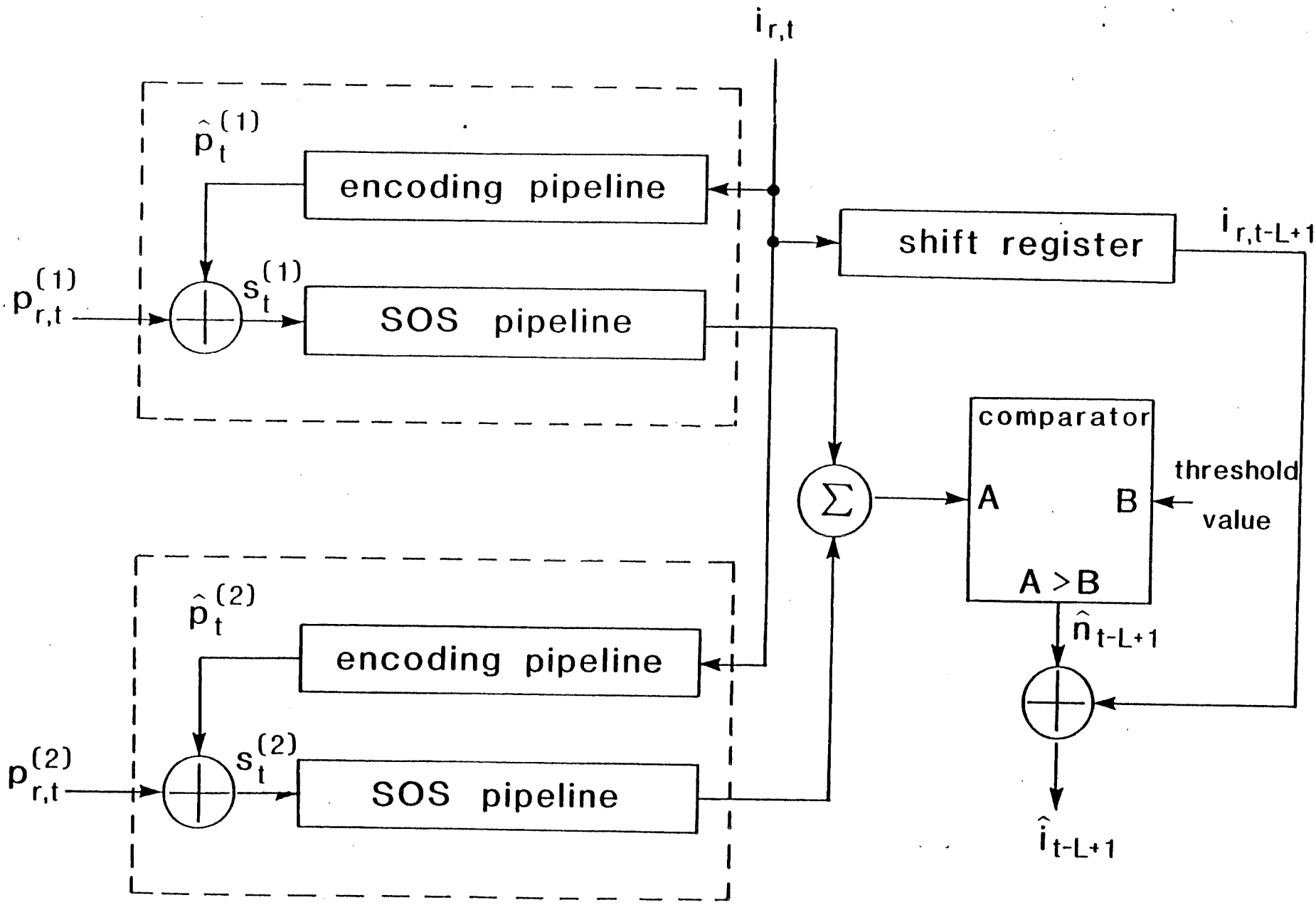


Fig. 12. An R=1/3 pipeline threshold decoder.

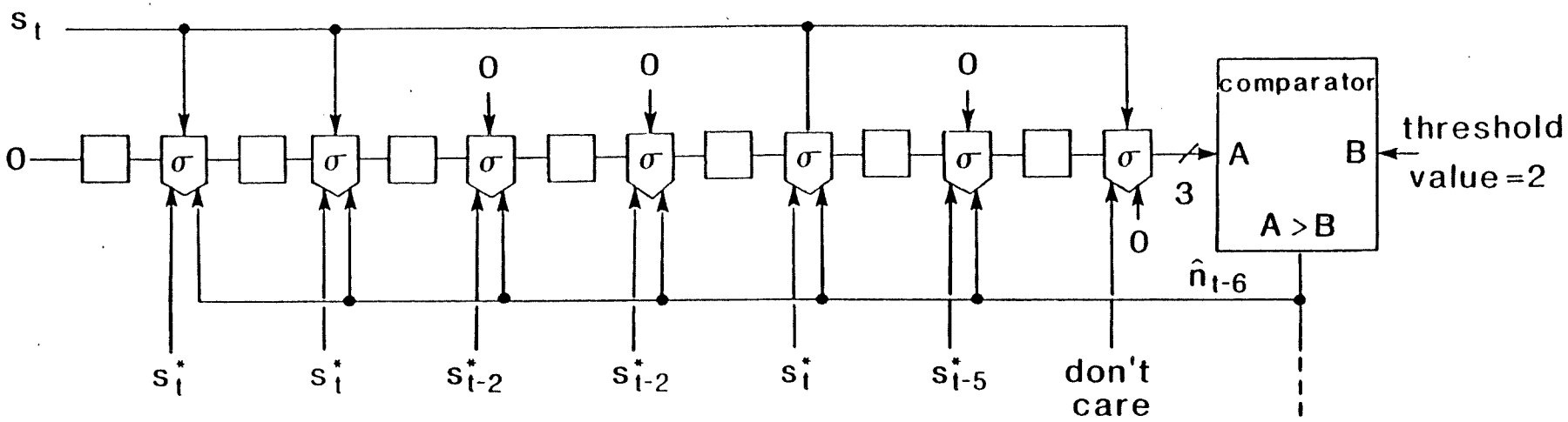
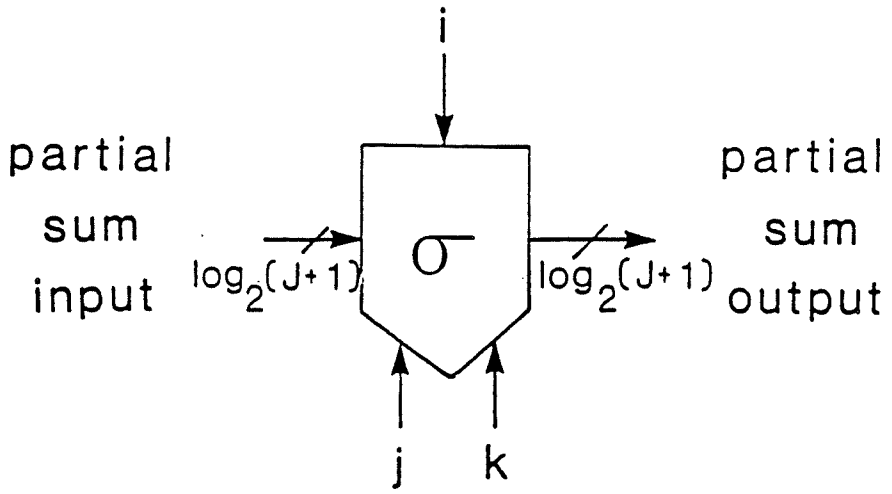


Fig. 13a. An SOS pipeline including feedback for the usual systematic rate 1/2, memory 6 code.



σ processor

Fig. 13b. One processor of the pipeline where i, j, and k are inputs for the connection, the target-syndrome value, and the noise estimate respectively.

i	j	k	σ
0	0	0	+0
0	0	1	+1
0	1	0	+0
0	1	1	−1
1	0	0	+1
1	0	1	+2
1	1	0	+1
1	1	1	+0

truth table

Fig. 13c. Truth table of the processor.

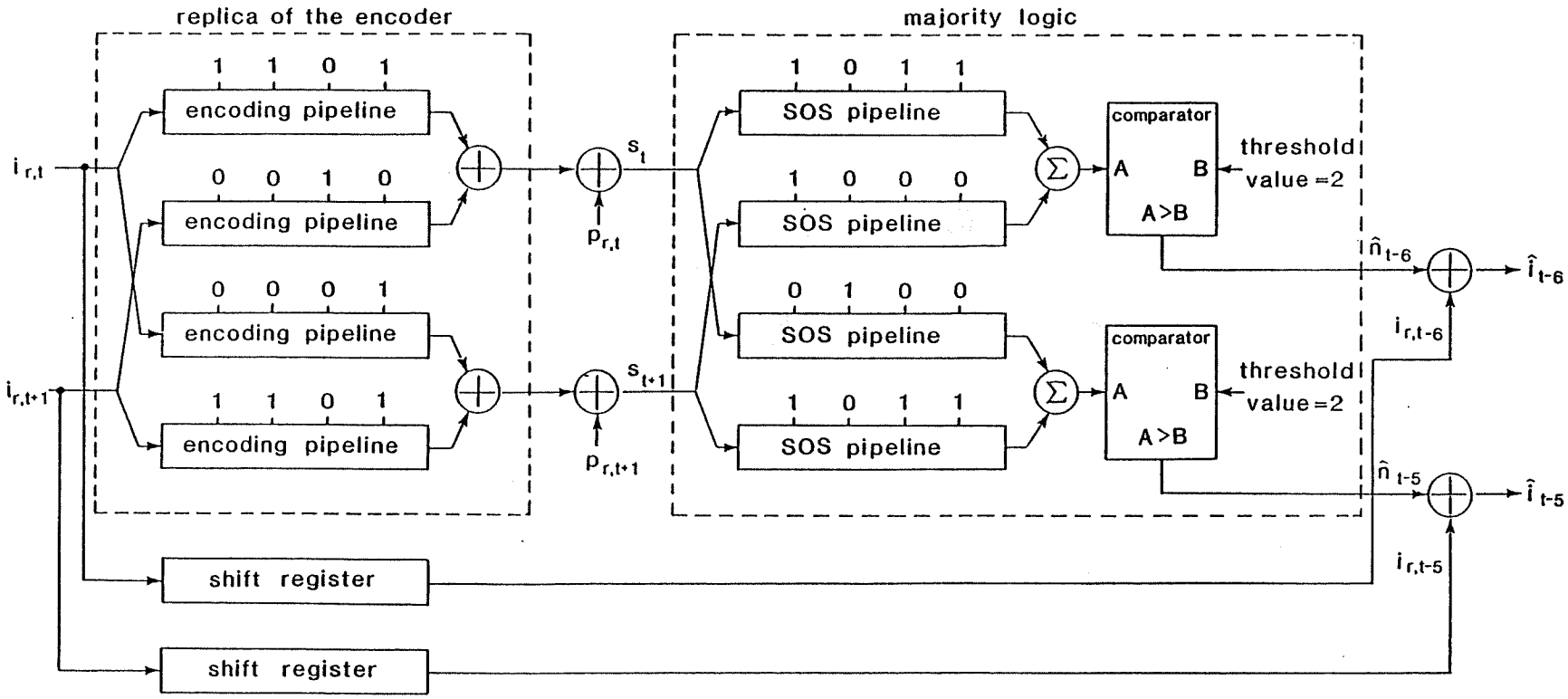


Fig. 14. An Y=2 parallel-pipeline threshold decoder for the usual rate 1/2, memory 6 code. The parallelism factor Y can be increased indefinitely, allowing even faster decoders.

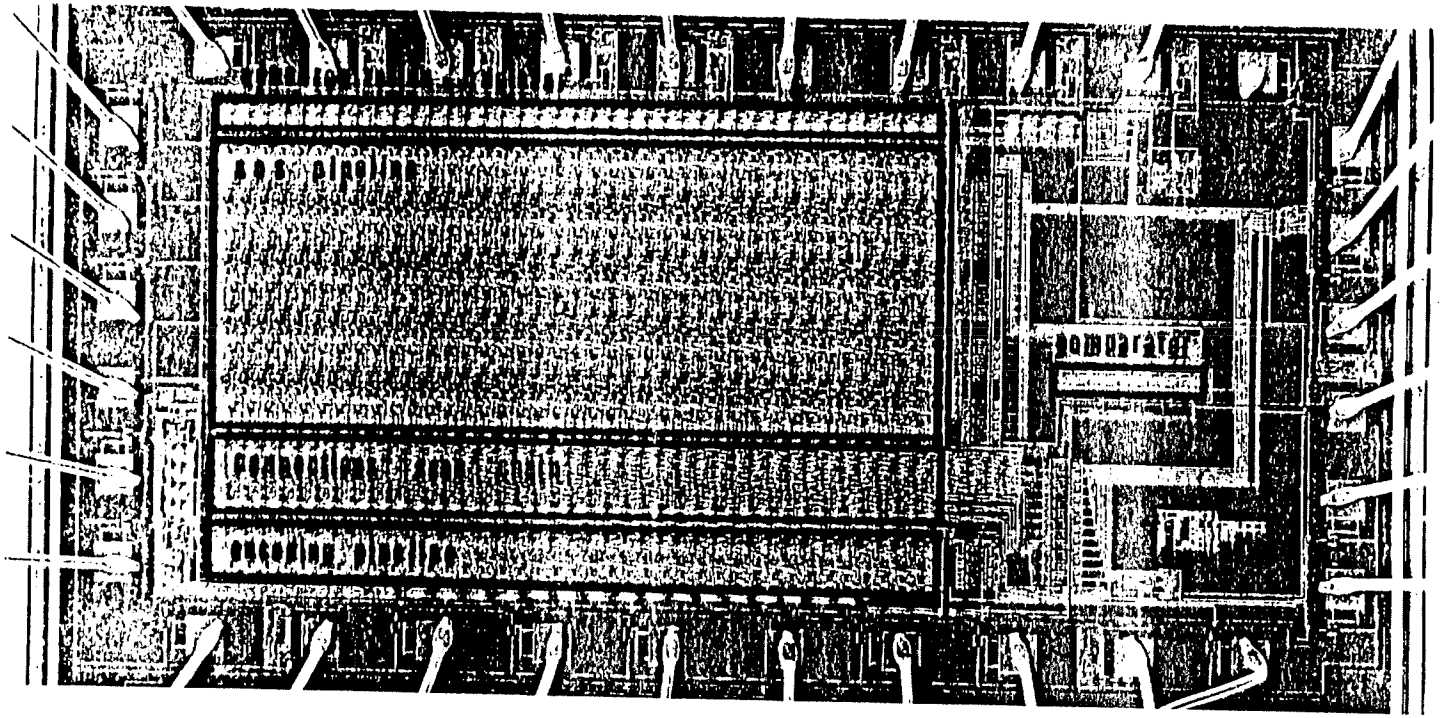


Fig. 15. Photomicrograph of an L=40 pipeline threshold decoder.

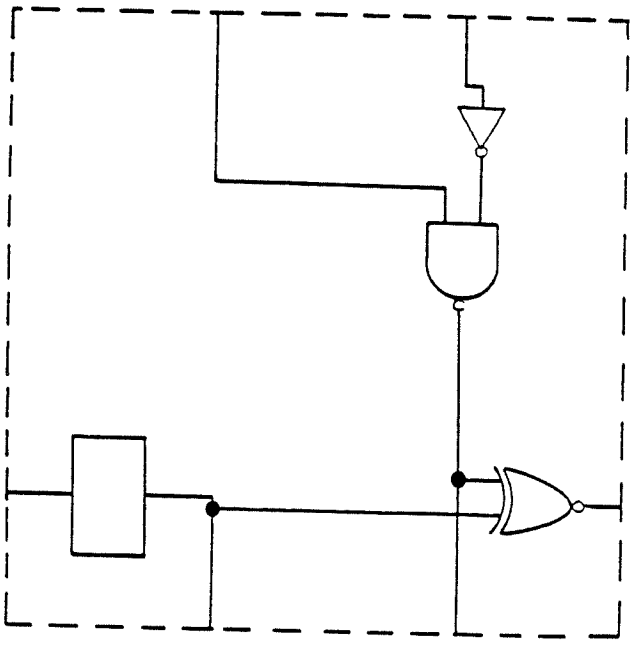


Fig. 16a. Logic diagram of one basic cell of the SOS pipeline.

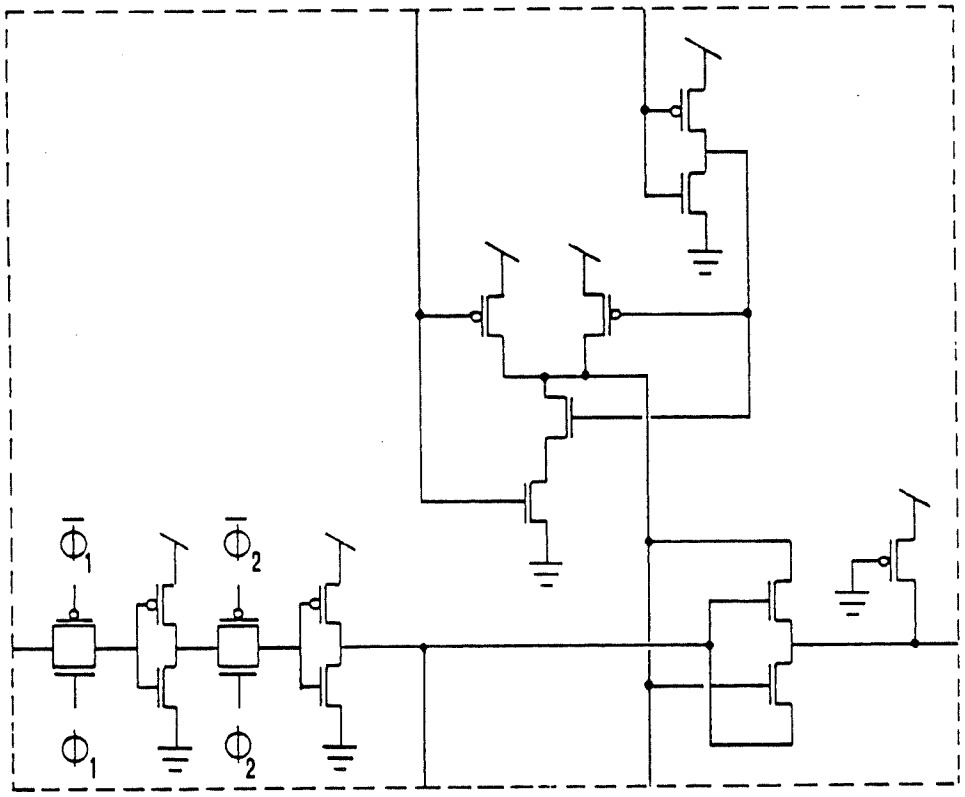


Fig. 16b. Transistor diagram of the same cell.